Access to LA Variables

Data Access Table Type

Thu, Jan 15, 2004

A local application allocates memory for housing variables during the time it is enabled for execution. Diagnostic information is often kept here to help the programmer analyze unusual events that are seen by the LA. But the location of this allocated memory changes every time the LA is restarted. This note describes two means of accessing such memory.

Data access table

The information needed to find a given variable from an LA's "static memory" is the name of the LA, plus the instance number, since it is possible to have more than one instance of the same LA enabled. The instance number is merely which one with the indicated name is found in scanning the LATBL from the beginning.

Suppose the following data access table format is used:

```
xx00   chan   LANAME
inst   offs   step   count
```

The LANAME is 4 characters, since the LOOP type name can be assumed. The instance number is 0, 1, 2, as needed. The offs indicates how deep the variable(s) of interest are within the LA static memory block. The step size is used when more than one variable is to be copied into the data pool. The count is the number of channels of data to produce.

To execute this new type, the LATBL must be searched for a match on the name and instance. Then the base address of the LA static memory is found, if any, and the offset applied to it to come up with the first, or only, variable to be accessed. One may assume that only 16-bit words are copied to keep it simple. Copy the word(s) as indicated by the step size. It may be useful to think about planning for this ahead of time during LA development.

It is more common to access LA variables by having the code in the LA copy the relevant data into the data pool itself. In that case, the target channel may be found among the LA instance parameter values in the LATBL entry. The scheme herein described can be used to remove the necessity of taking up parameters for this purpose, at the cost of making the targeted channels more obscure, as one may have to scan the RDATA table to find it.

Since one does not often move LATBL entries around, it may be sufficient to merely specify the LATBL entry number in the structure as follows:

```
xx00   chan   0000   offs
0000   indx   step   count
```

With this layout, one does not have to search the LATBL for a match, as the indx field already specifies the matching entry.

New listype

One might also consider making a data request for such variables housed within the SM structure of an LA instance. Then the ident might look as follows:

```
node
LA-
NAME
inst
offs
```

This uses 5 words, which is too long for an Acnet SSDN. An alternative approach could assume that the assignment of the LATBL instance is fixed, so that only the LATBL entry number need be specified. Then the ident could be:

```
node
indx
offs
```

With only 3 words in the ident, this ident format can fit within an Acnet SSDN. If one set the offset flag bit in the SSDN, and specified an offs field of 0000, then including an offset in the RETDAT request would permit reaching anywhere into the SM structure. One Acnet device could support access to any of the structure.

The internal ptr structure could include the LATBL entry number and the offset. The read-type routine would then look up the ptr to the SM structure from the LATBL entry and apply the offset to get the target address. If the LA is restarted, so that a new static memory block is allocated, the replies to an active request of this new listype would still find it. If the LA is not enabled, or there is no static memory allocated, then the reply data can be all zeros.

Settings

One can also permit settings to made to a LA static memory block. The ident, of course, is the same as for that to support data requests above. Errors can be returned in case there is no static memory block available for the indicated LATBL entry, or for targeting beyond the range of the static memory block. Needless to day, one must take care about implementing such settings. It is a tool that can really mess things up if used irresponsibly.

Post-implementation notes

The new Data Access type# that supports this feature is 0x33. Its format is as follows:

```
3300  chan    0000  offs
0000  indx    step  count
```

The indx is the index of the relevant LATBL entry. The offs is to be added to the base address of the static memory block to get the target field from which 16-bit words are to be copied into the target (ADATA) table entries starting at chan. The count is the number of words to be copied, and the step is the step size from one word to the next, if count > 1.

The new listype is #96, whose 3-word ident format is as follows:

```
node
indx
offs
```

Again, the indx is the LATBL entry number, and the offs is the offset into the targeted static memory block. The LA must be enabled and the static memory block allocated; otherwise, only zeros are returned, or, in the case of a setting, no setting action takes place.

Internally, the new ident-type index is 27, the read-type index is 31, the set-type index is 40, and the ptr-type index is 47. There is no limit on the number of bytes used for a setting. The internal ptr format has the offset to the LATBL entry in the hi word, and the offset into the static memory block in the lo word. Bit #30 is set to insure a non-null value. Error conditions detected by the ptr-type routine return an internal ptr of null, which causes the read-type routine to return zeros.